UV Distributed Information Management
Summer semester 2022

Assignment 01

Summary:

Deadline: April 20, 2022, 11:55 pm (aka 23:55) CET.

Extended Deadline: April 27, 2022, 11:55 pm (aka 23:55) CET.

Submission: Submit a compressed archive (e.g., a `zip` or a `tar.gz` file) that contains your Python3 code and the answers to the questionnaire via Blackboard.

Grading: 55% Python3 code, 45% answers (incl. meeting; cf. Section 5 for details).

---

1 General Remarks

The purpose of this assignment is to get in touch with a widely used general-purpose database management system (DBMS) named *PostgreSQL*. PostgreSQL is an open-source database system that is based on the relational data model [1]. It is rather easy to use and accessible using the Python3 programming language (i.e., in combination with the `psycopg2` [2] module).

Please submit your final Python3 code and the answers to the questionnaire until April 20, 2022, 11:55 pm (aka 23:55) CET via Blackboard [3] (late submission until April 27, 2022, 11:55 pm (aka 23:55) CET). Furthermore, please keep in mind that the exams contribute 46% to your final grade, hence you need to submit at least one assignment (partially) to pass the course.

1.1 Formatting Conventions

Commands (for the Linux and the `psql` command-line tool) and Python3 code are written in `TrueType` font. In addition, standalone `psql` terminal commands are written in bold font:

```
dbtutorial=# SELECT x FROM t WHERE a = b;
```

For commands that are to be executed in the `psql` terminal, the non-bold prefix `dbtutorial=#` denotes the database we are currently connected to (cf. Section 2.1). Linux terminal commands are formatted in a similar manner, i.e., the command is written in bold and the user that executes the command is the non-bold prefix:

```
root@database-tutorial:~# ls -lah
```

In the case of commands that are to be executed in the Linux terminal, the non-bold prefix `root@database-tutorial:~#` also includes the name of the machine we are currently working on (i.e., `database-tutorial`) as well as the directory we are currently in (i.e., `~` denotes the `home` directory of the current user). More information can be found in the supplementary material given in Section 4. Python3 code is simply wrapped in a box.

[1] `https://en.wikipedia.org/wiki/Relational_model`
[2] `https://www.psycopg.org/` and `https://pypi.org/project/psycopg2/`
[3] `https://elearn.sbg.ac.at`

1.2 Support

If you have troubles understanding the assignment, please use one of the following communication channels to get help (in this order):

1. **Lecture:** mondays 10:15 - 12:00 am CET, wednesdays 12:00 - 14:00 pm CET (exception: lecture-free periods).
2. **Slack:** `https://dbteaching.slack.com/archives/C01QQHPLFB2`.
3. **Email:** dkocher@cs.sbg.ac.at (as a last resort).

We recommend to start the assignment early. In case of a problem, it is easier for the instructor and other students to provide help in time if you identify the problem early.

Remark: Please notify the instructor as soon as possible if the assignment description is (partially) unclear or if there is a problem with the submission.

2 Assignment Description

We *highly recommend* that you read this section (including all the subsections) to the end before you start working (this should be less error-prone).

This assignment can be divided in five parts:

1. Setting up PostgreSQL (PSQL), cf. Section 2.1.
2. Create tables and fill tables with data, cf. Section 2.2.
3. Familiarize yourself (i.e., play around) with PostgreSQL, cf. Section 2.3.
4. Write an example application that accesses the database in Python, cf. Section 2.4.
5. Answer the questionnaire, cf. Section 2.5.

Only parts 1 (depending on your choice), 4 and 5 contribute to the overall grade of this assignment. We recommend to follow these steps and refer to the corresponding sections for details.

2.1 PostgreSQL Setup

In this section, we summarize two possible ways to set up PostgreSQL such that the assignment can be solved properly:

PostgreSQL Installation Install PostgreSQL on your own system (or virtual machine) using an operating system of your choice. This implies that you "pollute" your system with this installation and that the instructor may only be partially helpful (we will still try to help, but we mostly work on Linux systems). Nonetheless, we would like to emphasize that it is an *excellent exercise* for students to perform the installation of such a system on their own (at least once).

PostgreSQL Configuration in a VM Configure a pre-installed PostgreSQL in a virtual machine (VM) that runs Debian Linux. This implies that you will not experience the process of installing PostgreSQL on your own, but the instructor can more helpful since the VM runs Debian Linux. On the one hand, it may be cumbersome if you do not have any experience with Linux systems (and VMs), but on the other hand it may also be a nice opportunity to familiarize yourself and play around with a Linux system (and a VM).

You can choose your preferred way and the choice itself will not influence your grade in any way. However, make sure that you can answer questions regarding your choice.

2.1.1 PostgreSQL Installation

The first step is to install a PostgreSQL server locally on your (possibly virtual) machine. This should be possible on almost all operating systems, but the specific steps may diverge for different operating systems. Once PostgreSQL is installed, you can create a database and import data. Sources on how to install PostgreSQL can be found in the supplementary material provided in Section 4. Although PostgreSQL also provides a graphical user interface named *pgAdmin*, we recommend to also install PostgreSQL's *Command-Line Tools* (on some systems, they must be installed separately), which allow you to interact with the database system through the command line. In particular, the `psql` command-line tool (aka `psql` *terminal* or `psql` *shell*) provides the most basic way to use the database. Contrarily, the *Stack Builder* is not required for this assignment.

During installation, PostgreSQL also creates a default database named `postgres`. Open the `psql` command-line tool to connect to it (indicated by the `postgres=#` in the command-line tool). If you are asked for a server, a database, a port, or credentials (username + password), use the following default configuration:

- Server: `localhost` (or leave blank)
- Database: `postgres` (or leave blank)
- Port: `5432` (or leave blank)
- Username: `postgres` (or leave blank)
- Password: The password you set during the installation.

All available databases can be viewed by executing the `\l` command (mind the backslash) in the `psql` command-line tool. If you are using a fresh PostgreSQL installation, you should see three databases: `postgres`, `template0`, and `template1`.

For this assignment, we create our own database named `assignment1`. This can be done by executing the following command in the `psql` command-line tool:

```
postgres=# CREATE DATABASE assignment1;
CREATE DATABASE
postgres=#
```

The `postgres=#` at the beginning should already be displayed in your `psql` shell and is not part of the command. Similarly, the `CREATE DATABASE` in the second line is not to be entered by you. This is meant to be a feedback for the user (you) indicating that the preceding `CREATE DATABASE` command has been successfully executed. Afterwards, the `\l` command should print an additional database named `assignment1`. As database for this assignment, please use `assignment1`. In order to use the database `assignment1` for this assignment, you must connect to it *after* creating it (the creation itself does not connect to the newly created database; indicated by the fact that `postgres=#` is still displayed in your `psql` command-line tool). Therefore, please use the command `\c assignment1` to connect to the new database. After the connection is established, the command-line tool should display `assignment1=#` (instead of `assignment1=#`).

Remark: The creation of a database can take some time, hence please wait for the command to finish.

2.1.2 PostgreSQL Configuration in a VM

We also provide you with a so-called *virtual machine image*, which consists of a pre-installed Debian Linux as operating system and a pre-installed PostgreSQL installation. In this case, you

may have to familiarize yourself with the concept of a virtual machine (VM) and how to host the corresponding image. Essentially, a virtual machine [4] is a software that is designed to provide you with a substitute of a real machine, that is, it runs another operating system on top of your regular operating system – it *virtualizes* another system. For our assignment, it suffices to install a software named VirtualBox [5] that acts as a host for virtual machines. After you successfully installed VirtualBox, a so-called *image* can be used to set up the virtual machine that can be used for this assignment. The image (as `ova` file) can be downloaded from our Nextcloud [6] and runs Debian Linux [7] as operating system with an installation of PostgreSQL.

After downloading the `debian-postgresql-vm.ova` file, this file must be imported into VirtualBox. There are many online tutorials [8] on how to accomplish this and it primarily consists of two steps: (1) Choosing the image to import (navigate to the downloaded file) and (2) checking/modifying the settings of the virtual machine (typically no modifications are necessary, but sometimes, for example, you may have to disable the USB port). Once you click on the *Import* button, VirtualBox should import the image and set up the virtual machine that can be used for this assignment. VirtualBox then shows a new VM on the left-hand side, which can be started/booted with a double-click.

For this VM, there exist two users: (1) `dbtutorial` and (2) `root`. If you are not familiar with the concept of a `root` user (or superuser), please check out the corresponding article [9] on Wikipedia. In essence, the `root` user has all privileges whereas the `dbtutorial` user has not. By default, we will work with the `dbtutorial` user, but we will temporarily switch to the `root` user if we need additional privileges (e.g., to install a software package). The password for both users is the same: `dbpwd1`

Once you logged into the VM and before you start working on the actual assignment, try to familiarize yourself a bit with Debian Linux (unless you already have experience using Linux systems). For example, open a *terminal* (aka *shell* or *command-line tool*) and try some basic commands (some links can be found in the supplementary material, cf. Section 4). In particular, we will use the `psql` command-line tool (aka *PSQL terminal* or *PSQL shell*), which provides the most basic way to use the database.

Before we can use the database, we need to create a database user named `dbtutorial`. Note that there are two types of users, users of the operating system and users of PostgreSQL/the database. First, we switch to the system's `root` user:

```
dbtutorial@database-tutorial:~$ su -
Password:
root@database-tutorial:~#
```

The `dbtutorial@database-tutorial:~$` (as well as `root@database-tutorial:~#`) at the beginning should already be displayed in your terminal and is not part of the command itself (which is `su -`). Then, we switch to the system's `postgres` user (which also exists in our database) and connect to the default database named `postgres` (indicated by the `postgres=#` in the command-line tool):

```
root@database-tutorial:~# su postgres
postgres@database-tutorial:/root$ psql
postgres=#
```

Now, we are in the `psql` shell, i.e., we can directly send commands to the database. All available databases can be viewed by executing the `\l` command (mind the backslash) in the `psql`

[4] https://en.wikipedia.org/wiki/Virtual_machine
[5] https://www.virtualbox.org/manual/ch02.html
[6] https://kitten.cosy.sbg.ac.at/index.php/s/H7P2qEg5iZiTdXd
[7] https://en.wikipedia.org/wiki/Debian
[8] https://docs.oracle.com/cd/E26217_01/E26796/html/qs-import-vm.html
[9] https://en.wikipedia.org/wiki/Superuser

command-line tool. You should see three databases: `postgres`, `template0`, and `template1`. Next, we create a new user [10]:

```
postgres=# CREATE USER dbtutorial WITH CREATEDB LOGIN PASSWORD 'dbpwd1';
CREATE USER
postgres=#
```

Once the database's `dbtutorial` user exists, we can switch back to the system's `dbtutorial` user and can start using PostgreSQL with our newly created user (still on the default database, indicated by `-d postgres`):

```
postgres=# \q
postgres@database-tutorial:/root$ exit
exit
root@database-tutorial:~# exit
logout
dbtutorial@database-tutorial:~$ psql -d postgres
postgres=>
```

For this assignment, we create our own database named `assignment1`. This can be done by executing the following command in the `psql` command-line tool:

```
postgres=> CREATE DATABASE assignment1;
CREATE DATABASE
postgres=>
```

Again, the `postgres=>` at the beginning should already be displayed in your `psql` shell and is not part of the command. Similarly, the `CREATE DATABASE` in the second line is not to be entered by you. This is meant to be a feedback for the user (you) indicating that the preceding `CREATE DATABASE` command has been successfully executed. Afterwards, the `\l` command should print an additional database named `assignment1`. As database for this assignment, please use `assignment1`. In order to use the database `assignment1` for this assignment, you must connect to it *after* creating it (the creation itself does not connect to the newly created database; indicated by the fact that `postgres=>` is still displayed in your `psql` command-line tool). Therefore, please use the command `\c assignment1` to connect to the new database. After the connection is established, the command-line tool should display `assignment1=>` (instead of `postgres=>`).

Remark: The creation of a database can take some time, hence please wait for the command to finish.

2.2 Data Initialization

Once your database is ready, the database contains no data, i.e., it is empty. By executing the `\d` command, the database will report that no relations (tables) have been found. Therefore, we first have to populate the database with some data.

In the course of this assignment, we will use parts of the publicly available Internet Movie Database (IMDB) [11] . First, we have to download the data from our Nextcloud [12]. Note that you may not be able to use the typical `CTRL+C` and `CTRL+V` sequence with your virtual machine since the clipboard is not shared between the "real" system and the virtual machine by default [13]. Second, we use the data definition language (DDL) of PostgreSQL to create the tables that will store the data. Finally, we fill the tables with data. The data will be provided through two plain files that are then imported such that their contents fill the respective tables.

[10] `https://www.postgresql.org/docs/current/sql-createuser.html`
[11] `https://www.imdb.com/interfaces/`
[12] `https://kitten.cosy.sbg.ac.at/index.php/s/HEKwezj4PfQSA3T`
[13] `https://www.youtube.com/watch?v=fqrJ7qlhJu0`

We provide an SQL script (`create_db.sql`) that contains the statements that are required to create the tables. There are two possible ways to execute the statements. Either you execute the script directly from the `psql` command-line tool (using `\i create_db.sql`), or you copy and execute each statement separately using the `psql` command-line tool:

```
assignment1=# \i /path/to/imdb/folder/create_db.sql
CREATE TABLE
CREATE TABLE
assignment1=#
```

To fill the tables with data, we provide you with two plain files and recommend to execute the following two commands (one after another; mind the semi-colon at the end) in your `psql` command-line tool:

```
assignment1=# \COPY titles FROM title.basics_no_header_array_format.tsv WITH DELIMITER E'\t';
COPY 5447872
assignment1=# \COPY names FROM name.basics_no_header_array_format.tsv WITH DELIMITER E'\t';
COPY 8991013
```

Remark: Importing the files into the tables will take some time, hence please wait for the commands to finish.

On some systems, the encoding must be specified explicitly when importing the data from file. If you run into problems when importing the files, please try the following commands:

```
assignment1=# \COPY titles FROM title.basics_no_header_array_format.tsv WITH DELIMITER E'\t' ENCODING 'UTF8';
COPY 5447872
assignment1=# \COPY names FROM name.basics_no_header_array_format.tsv WITH DELIMITER E'\t' ENCODING 'UTF8';
COPY 8991013
```

After the import, PostgreSQL provides feedback on the number of tuples that have been imported, i.e., 5,447,872 and 8,991,013 tuples have been imported into the tables `titles` and `names`, respectively. Afterwards, you can see all available tables by executing `\d`. There should be two tables in the database: `names` (of actors) and `titles` (of movies). You can also look at the schema of a particular table. For example, the command `\d names` shows the schema (and some additional information) of the table `names`.

2.3 PSQL Introduction

After the successful import of the data, try to further familiarize yourself with the `psql` command-line tool. We provide the following SQL queries that can be executed out of the box by entering them into the `psql` command-line tool (one after another; mind the trailing semi-colon):

Query Q1:

```
assignment1=# SELECT * FROM names WHERE primaryName = 'Chris Hemsworth';
```

Query Q2:

```
assignment1=# SELECT * FROM titles WHERE primaryTitle = 'The Avengers' AND titleType = 'movie';
```

Query Q3:

```
assignment1=# SELECT primaryTitle FROM names, titles
WHERE names.birthYear = titles.startYear AND names.primaryName = 'Scarlett Johansson';
```

Query Q4:

```
assignment1=# EXPLAIN SELECT * FROM titles WHERE primaryTitle = 'The Avengers' and titleType = 'movie';
```

SELECT, FROM, and WHERE clauses have briefly been covered in the lecture. The * in the SELECT clause specifies that *all* columns of the table in the FROM are retrieved.

In query Q2, the WHERE clause consists of two conditions that are linked using the AND operation. This means that every tuple that is part of the result must satisfy both conditions. For example, a TV series that is named "The Avengers" is not found because it is not a movie.

Query Q3 *joins* the two tables based on the condition in the WHERE clause. A join links tuples of two (or more) tables. In this specific case, we link the year of birth of the actors (`names.birthYear`) with the year of publication of the movies (`titles.startYear`). Furthermore, we specify that only actors with the name "Scarlett Johansson" should be considered. Intuitively, this means we ask the database system to show all movies that started in the year of birth of Scarlett Johansson. In this case, the FROM clause contains two tables that are separated by a comma. Without the WHERE clause, this results in the Cartesian product, i.e., every row of the first table (names) would be linked with every row in the second table (titles). This can result in a very large result, therefore we restrict the join using the conditions in the WHERE clause.

The last query, Q4, is basically Q2 with a small extension: We put the EXPLAIN keyword [14] in front of the query. EXPLAIN shows the so-called *query plan*, that is, information about the steps PostgreSQL plans to execute in order to determine the result (without executing the actual query). As an exercise (not part of this assignment), the other queries could also be extended with the EXPLAIN keyword.

2.4 Access the Data Using Python3

Although the psql command-line tool can be used to execute queries, a database system is typically accessed by an application. Therefore, the fourth part of this assignment is to write a small Python3 application. We recommend to use the psycopg2 module (or driver) for Python3 to (a) establish a connection to your local database, (b) execute the queries and retrieve the results, and (c) close the connection to your local database. Your application should execute the queries Q1, Q2, and Q4 as provided in Section 2.3 and print the respective results (i.e., all the tuples that are returned by PostgreSQL; output format does not matter as long as it is human-readable).

For query Q3, you are required to do a small modification: Q3 as given in Section 2.3 returns a list of tuples that satisfy the condition in the WHERE clause. In your Python3 application, Q3 should only return the *number* of tuples that satisfy the condition in the WHERE clause. This can either be accomplish by modifying the SQL command itself to count the number of tuples (hint: use the COUNT aggregate function [15]) or by adapting the Python3 code such that not all the tuples are printed but only the number of tuples (hint: use the `len()` function [16]).

Remark: Please do not confuse the psycopg2 module with the (relatively) new psycopg3 module for Python. We use to the psycopg2 module in combination with Python3.

Template Code There are many tutorials regarding the installation [17] and the usage of psycopg2 [18] in combination with PostgreSQL. Nonetheless, we provide a minimum template code in Python3 that can be used as a starting point.

[14] https://www.postgresql.org/docs/current/using-explain.html
[15] https://www.postgresql.org/docs/current/functions-aggregate.html
[16] https://docs.python.org/3/library/functions.html#len
[17] https://www.psycopg.org/ and https://pypi.org/project/psycopg2/
[18] https://wiki.postgresql.org/wiki/Psycopg2_Tutorial

Listing 1: Template code to access a database using psycopg2.

```
#!/usr/bin/python3

import psycopg2 as pg2

if __name__ == "__main__":
  try:
    # Establish a connection to the database 'assignment1' with user 'dkocher'
    # and password 'mypw'.
    connection = pg2.connect("dbname='assignment1' user='dkocher' password='mypw'")
  except:
    print("Unable to establish a connection to {}".format('assignment1'))

  # Retrieve a so-called cursor in order to interact with the database.
  cursor = connection.cursor()

  try:
    # Send a simple SELECT query to the database.
    cursor.execute("""SELECT * FROM names WHERE primaryName = 'Christian Bale'""")

    # Retrieve the entire result of the query.
    # cursor.fetchone() would retrieve only the first tuple of the result.
    records = cursor.fetchall()

    # Print the retrieved result.
    for record in records:
      print("{}".format(record))
  except:
    print("Unable to execute simple SELECT query.")
  finally: # The finally-branch is always executed (independently of an exception).
    if cursor is not None:
      # Close the cursor.
      cursor.close()

    if connection is not None:
      # Close the connection.
      connection.close()
```

2.5 Questionnaire

The questionnaire contains questions about the assignment. These questions are potentially discussed during the after-assignment meetings. The questionnaire can be found in a separate textfile called `assignment1-questionnaire.txt`.

3 Submission

Please submit a single compressed archive (e.g., `.zip` or `.tar.gz`) that contains two files: (a) The code of your Python3 application and (b) the answers to the questionnaire.

Code Please submit a single Python3 file (`.py`) that contains the full code for this assignment, i.e., the connection to the database and the execution of the four queries. The code must print the results of *all* four queries (one after another) when executed *as submitted*. We will not change your code, for example, change some variable to see the result of a certain query. Therefore, please double-check that *all* four queries are executed (and also that query Q3 is modified as described in Section 2.4).

Remark: Please consider removing the database credentials (i.e., username and password) before you submit your code (in case you did not use the default ones as described in Section 2.1).

Questionnaire You can answer the questions directly in the textfile `assignment1-questionnaire.txt`. If you prefer to use a different application to answer the questions (e.g., Microsoft Word and the likes), you are welcome to do so. In any case, the submitted file must be in one of the following formats: `.txt`, `.pdf`, `.odt`, `.doc`, or `.docx`.

Remark: The recommended formats are `.txt` and `.pdf`.

4 Supplementary Material

This section provides a list of pointers to material that may be helpful to solve the assignment.

- PostgreSQL: `https://www.postgresql.org/`
- Download PostgreSQL: `https://www.postgresql.org/download/`
- The full PostgreSQL documentation (in particular, Chapter 1 may be helpful): `https://www.postgresql.org/docs/current/`
- Other resources regarding installation and usage of PostgreSQL:
  - One of the many "Getting Started" guides: `https://www.youtube.com/watch?v=BLH3s5eTL4Y`
  - Installing PostgreSQL: `https://www.postgresqltutorial.com/install-postgresql/`
  - PostgreSQL on Windows: `https://www.postgresql.org/download/windows/`
- Documentation of the `EXPLAIN` statement: `https://www.postgresql.org/docs/current/using-explain.html`
- Python Modules: `https://docs.python.org/3/installing/index.html`
- The `psycopg2` module: `https://www.psycopg.org/`
- Installation of the `psycopg2` module for Python:
  - Official website: `https://www.psycopg.org/install`
  - The Python Package Index: `https://pypi.org/project/psycopg2/`
- The `psycopg2` tutorial in the PostgreSQL wiki: `https://wiki.postgresql.org/wiki/Psycopg2_Tutorial`
- One of the many "Getting Started" guides: `https://www.youtube.com/watch?v=2PDkXviEMD0`
- One of the many introductions to the Linux terminal: `https://www.digitalocean.com/community/tutorials/an-introduction-to-the-linux-terminal`

5 Grading

For the sake of transparency, this section provides more details on the grading of this assignment, i.e., which part contributes how many points to the total number of 18 points for this assignment.

Code The code contributes at most 10 points and is evaluated based on the following criteria (if the code is executed as submitted; disregarding the credentials):

Max. Points	Criterion
2	Q1 is executed and the correct result is printed to the command line.
2	Q2 is executed and the correct result is printed to the command line.
2	Modified Q3 is executed and the (correct) result is printed to the command line.
2	Q4 is executed and the (correct) result is printed to the command line.
2	Answer two questions regarding your code in the after-assignment meeting.
10	

Questionnaire The questionnaire contributes at most 8 points and is evaluated based on the following criteria (taking the discussion in the after-assignment into account):

Max. Points	Criterion
2	Correctness of answer A1.
2	Correctness of answer A2.
2	Correctness of answer A3.
2	Correctness of answer A4.
8	